8-Ч

УДК 004.4’42

Ю.В. Чернухин, М.Ю. Поленов, Д.В. Левченко
Таганрогский технологический институт Южного федерального университета, Россия
chernukhin@dce.tsure.ru, polenov@dce.tsure.ru, d.l@list.ru

Возможности применения среды мультитрансляции в качестве компилятора компиляторов

Рассматривается подход к применению разработанной ранее среды многоязыковой трансляции моделей – Мультитранслятора для конверсии программных проектов с одних языков программирования высокого уровня в проекты на других языках. Развитие данного подхода позволяет использовать Мультитранслятор в качестве среды для синтеза частных трансляторов в режиме компилятора компиляторов.

Введение

В настоящее время активно проводятся исследования в области разработки инструментальных средств синтеза трансляторов. В основном все существующие средства автоматизации трансляции относятся к классу компилятора компиляторов (compiler-compiler) [1], которые работают по принципу генерации исходного кода конечного (специализированного) транслятора, называемого также частным транслятором, на основе описанных пользователем грамматик. В качестве языка для генерации исходного кода используется один из широко известных языков [2]: C, C++, C#, Java.

Проведенный анализ существующих средств синтеза трансляторов показал, что данные средства не предоставляют возможности интегрированного описания частей систем продукций и ориентированы на использование в цикле генерации кода частного транслятора сторонних компиляторов. Поскольку пользователю не предоставляется единая среда разработки, то отлаживать и исправлять отдельные части транслятора приходится при помощи различных программных оболочек.

Перечисленные проблемы усложняют процесс синтеза и отладки трансляторов и делают актуальной задачу разработки средств многоязыковой трансляции, которые позволят устранить отмеченные недостатки. К данным средствам предъявляются следующие требования:

– наличие средств редактирования и отладки грамматик;
– наличие возможности задавать действия непосредственно в правилах перевода;
– отсутствие необходимости использования дополнительных сторонних компиляторов;
– наличие интегрированной среды для выполнения всего процесса разработки, отладки и проверки грамматик и генерации кода транслятора в пределах одной программной оболочки;
– отсутствие требований к пользователю изучать дополнительные языки и технологии программирования.

Всем перечисленным требованиям удовлетворяют развиваемый авторами подход к организации инструментальных средств многоязыковой трансляции и разработанная в рамках данного подхода единая транслирующая среда – Мультитранслятор [3].

Мультитранслятор (МТ) [4] был изначально разработан для обеспечения трансляции внешних моделей для сред виртуального моделирования сложных систем. Экспериментальные исследования МТ в качестве подсистемы импорта моделей среды

VTB [5] показали его эффективность при конверсии программных моделей. Данные исследования также показали, что заложенные в Мультитрансляторе принципы использования универсального ядра трансляции как при разработке трансляционных модулей, так и при переводе моделей, позволяют применять МТ в качестве инструментальной среды разработки трансляторов (компилятора компиляторов).

1. Подходы к решению проблемы синтеза частных трансляторов

Процесс разработки частного транслятора при помощи современных средств построения трансляторов традиционно сопровождается использованием дополнительных компиляторов и языков программирования. Укрупненно данный процесс можно представить в виде схемы, изображенной на рис. 1.

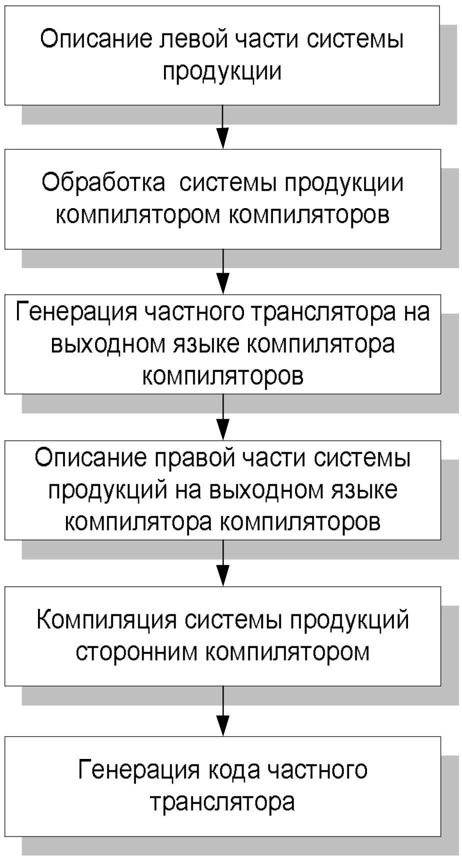


Рисунок 1 – Организация работы компилятора компиляторов при синтезе частного транслятора

Как видно из рис. 1, при создании частного транслятора при помощи компилятора компиляторов процесс описания продукционных правил разделяется на три этапа. На первом этапе вводятся левые части правил, описывающие грамматику входного языка, на втором этапе данная грамматика обрабатывается компилятором компиляторов, и лишь третий этап посвящен определению пользователем правой действенной части, завершающей формирование системы продукций. При этом возникает ряд рассмотренных выше проблем, снижающих эффективность этого процесса.

Предлагаемый подход к решению данных проблем основан на применении Мультитранслятора в качестве компилятора компиляторов для создания частных трансляторов. Использование МТ для этих целей позволяет значительно сократить количество промежуточных этапов разработки частного транслятора, уменьшить время разработки такого транслятора, а также избежать необходимости подключения внешнего компилятора для генерации его кода.

Мультитранслятор [3] позволяет создавать трансляционные модули по переводу исходных текстов, написанных на одних языках программирования, в тексты на других языках. В состав МТ входят: компилятор, работающий со специальным языком – языком описания грамматик; отладчик, с помощью которого можно проверить работоспособность написанных грамматик; многофункциональная интегрированная оболочка, предоставляющая пользователю широкий набор возможностей редактирования, управления проектами, компиляции и отладки.

Процесс разработки частного транслятора при помощи Мультитранслятора можно представить в виде схемы, изображенной на рис. 2.



Рисунок 2 – Организация работы Мультитранслятора при синтезе частного транслятора

В задачу пользователя Мультитранслятора входит составление трансляционного модуля в виде системы продукций, состоящей из грамматических правил входного языка и набора соответствующих этим правилам действий для генерации выходной программы.

Как видно из рис. 2, описание левых и правых частей правил выполняется на одном этапе проектирования, и помимо этого у пользователя появляется предоставляемая МТ дополнительная возможность отладки и тестирования полной системы продукций в процессе разработки.

2. Использование Мультитранслятора в режиме компилятора компиляторов для создания частного транслятора с языка Delphi на язык C++ Builder

В рамках рассматриваемого подхода к использованию Мультитранслятора как компилятора компиляторов и построению с его помощью частных трансляторов, в среде МТ был разработан транслятор для перевода проектов с языка Delphi [6] на язык C++ Builder [7].

В настоящее время языки C++ и C# занимают лидирующее положение среди универсальных алгоритмических языков программирования. Доля языка Pascal и его развития – Delphi в последнее время снижается, а популярность новых языков, таких, как C#, растет. В связи с этим актуальна задача перехода на более современные языковые средства разработки. Основными проблемами при таком переходе являются перевод исходных кодов проектов на новый язык программирования, а также переобучение команды разработчиков. Следовательно, целесообразно при этом осуществлять плавный переход на среду разработки, которая не отличается коренным образом от используемой в настоящий момент. В случае с языком Delphi логичным решением будет переход на среду разработки Borland C++ Builder, поскольку они имеют схожий интерфейс и одинаковые библиотеки стандартных классов.

Для синтеза транслятора с Delphi на C++ Builder рассмотрим особенности данных языков.

2.1. Отличительные особенности входного и выходного языков трансляции

Языки Delphi и C++ Builder находятся на одном уровне по функциональности и имеют схожие грамматические конструкции. Однако некоторые конструкции, в силу различных объектных моделей языков, требуют повышенного внимания при трансляции. Рассмотрим подробнее эти конструкции и связанные с ними особенности перевода программ.

Наиболее существенным отличием является объектная модель и способы работы с памятью в Delphi и C++ Builder. В Delphi объекты всегда создаются в куче и для обращения к ним используется оператор членства «.». Этот оператор используется также для обращения к полям структур. В C++ объекты могут создаваться как в куче, так и в стеке. Если они создаются в куче, то доступ к ним осуществляется с помощью оператора косвенного членства «–>», а доступ к структурам и объектам, созданным в стеке, осуществляется с помощью оператора прямого членства «.». Это означает, что при генерации кода необходимо выбрать правильный оператор для обращения к объекту, а также необходимо вести учет идентификаторов и их типов.

Другим отличием объектной модели являются требования к структуре классов. Классы, созданные в Delphi, могут не содержать конструктора, используя конструктор предка по умолчанию. В программе на C++ Builder инициализация объекта должна выполняться при помощи конструктора текущего класса. Транслятор должен автоматически вставлять конструктор для тех классов, где он явно не объявлен.

Вызов базового метода в Delphi осуществляется при помощи ключевого слова *inherited*, причем если параметры не указаны, то автоматически передаются параметры, полученные текущим методом. В языке C++ необходимо указывать также имя базового класса, чей метод должен быть вызван, и вручную передавать параметры. Кроме того, отличается описание вызова конструктора базового класса, а деструктор базового класса в C++ Builder вызывается автоматически. Транслятор также должен учитывать все эти особенности.

Также существенным отличием языков является регистрозависимость имен идентификаторов и констант. Если в программе на Delphi идентификатор может быть объявлен строчными буквами, а использоваться в теле программы записанным прописными буквами, то с точки зрения C++ это два разных идентификатора. Подобные проблемы возникают и с именами функций.

Все рассмотренные отличия в представлениях проектов на входном языке Delphi и выходном языке C++ Builder были учтены при синтезе трансляционного модуля Мультитранслятора, являющегося основой создаваемого частного транслятора для данной пары языков. Рассмотрим основные аспекты разработки этого трансляционного модуля в среде Мультитранслятора.

2.2. Синтез трансляционного модуля Мультитранслятора для частного транслятора с языка Delphi на язык C++

Создание трансляционного модуля МТ начинается с формирования главной процедуры *main*. Данная процедура описывает основные этапы работы разрабатываемого модуля частного транслятора. В приведенном фрагменте главной процедуры трансляционного модуля с Delphi на C++ данные этапы можно выделить по комментариям в тексте процедуры:

```
int main(){
     print("Запуск транслятора.\n");
     // Обработка входных параметров
     if (!ProcessInputParams()){
          print("Ошибка при разборе входных параметров\n");
          return 1;}
     // Инициализация трансляционного модуля
     TranslatorInit();
     // Подготовка рез. каталога (копирование шаблонов и файлов)
     PrepareTemplates();
     // Анализ и разбор dpr файла и получение списка .pas файлов
     if (!AnalyzeProjectFile(aInputFullPath)){
          print("Ошибка при разборе конф. входного проекта\n");
          return 1;}
     // Трансляция конфигурационных файлов проекта
     CreateProjectFiles();
     print("Трансляция конф. файлов проекта завершена\n\n");
     // Трансляция всех найденных файлов
     for (int i = 1; i <= nUnitCount){
          UnitGetInfoByIndex(i, curUnit);
          // Если модуль не системный, то он транслируется
          if (curUnit.Form != ""){
               // Подготовка файлов для текущего модуля
               PrepareFiles();
               print("Анализ модуля " + aCurrentFilePath + "\n");
               if (!Parse("Goal")){
                     print("Ошибка при трансляции модуля"+
                    aCurrentFilePath +"\n");
                    return ;}
               print("Трансляция модуля завершена успешно \n");
               // Сохранение сгенерированных файлов
               SaveFiles();
               print("\n");
               }
      }
     next(i++);
     // Удаление временных файлов
     FileRemove(aTempFilePath);
     FileRemove(aTempFilePath + ".imd");
     print("Трансляция проекта завершена\n");
     return 0;}
```

Поскольку при трансляции программ с Delphi на С++ возникает проблема перевода полных проектов, целесообразно представить главную процедуру трансляционного модуля в виде совокупности процедур и функций. Для рассматриваемого трансляционного модуля были реализованы следующие процедуры: разбора входных параметров *ProcessInputParams()*; инициализации транслятора *TranslatorInit()*; подготовки шаблонов *PrepareTemplates()*; анализа файла проекта *AnalyzeProjectFile()*; создания файлов проекта *CreateProjectFiles()*; подготовки файлов для текущего модуля *PrepareFiles()*; сохранения файлов, сгенерированных при трансляции *SaveFiles()*. Для трансляции программ использовалась стандартная функция *Parse()* Мультитранслятора.

После разработки главной процедуры реализуются продукционные правила, описывающие перевод конструкций входного языка на выходной язык. Рассмотрим фрагмент трансляционного модуля с Delphi на С++, описывающего структуру программного модуля на Delphi на языке описания грамматик Мультитранслятора:

```
rule <"Unit">:error = "Incorrect declaration",{
   variant{
     symbol "unit": error = "'unit' word is absent"{}
     symbol STD_ID{
          // Фиксировать имя модуля для генерации заголовков.
          CurrentModule.Name = GetText();}
     symbol ";"{}
     symbol ""{
          // Сформировать заголовок .cpp файла
          aLocalTemp = StrReplace(aCPPTemplate,
               aModuleHeaderName_t, CurrentModule.Name + ".h", true);
          // Переключиться на .cpp файл
          SetActiveFile(0);
          // Записать заголовок .cpp файла
          MemWrite(aLocalTemp);
          // Сформировать заголовок .h файла
          aLocalTemp = StrReplace(aHStartTemplate, aHeaderName_t,
            CurrentModule.Name + "H", true);
          // Переключиться на .h файл
          SetActiveFile(1);
          // Записать заголовок .h файла
          MemWrite(aLocalTemp);
          // Перейти в интерфейсную секцию.
          Place.UnitPosition = psInterface;}
     symbol <"InterfaceSection">{
          aVarSectionDecl = "";}
     symbol ""{
          Place.UnitPosition = psImplementation;
          // Переключиться на .cpp файл
          SetActiveFile(0);}
     symbol <"ImplementationSection"> {
          // Переключиться на .h файл.
          SetActiveFile(1);
          // Дописать конец заголовка
          MemWrite(aHEndTemplate);}
     symbol ""{
          Place.UnitPosition = psInitialization;}
     symbol [<"InitSection">]     {}
     symbol ".": error = "'.' symbol is absent" {}
        }
     }
```

Аналогично в трансляционном модуле реализуются остальные языковые конструкции Delphi, сопровождающиеся соответствующими действиями по генерации выходного кода.

2.3. Реализация частного транслятора

Путем подключения разработанного трансляционного модуля *Delphi2C++Builder* к ядру МТ был создан частный транслятор, представляющий собой проект, загружаемый и выполняемый интегрированной средой Мультитранслятора.

Для реализации частного транслятора в виде приложения использовалась функция Мультитранслятора, позволяющая компилировать собственные проекты в консольный исполняемый файл, использующий при работе только библиотеки ядра трансляции и стандартных функций. Таким образом, было получено независимое консольное приложение, которое в дальнейшем может быть использовано без оболочки Мультитранслятора при трансляции проектов на Delphi в проекты C++ Builder.

Как уже отмечалось выше, в работе созданного на базе МТ частного транслятора, выполняющего трансляцию проектов на Delphi, можно выделить следующие этапы: инициализация, анализ конфигурационных файлов, выделение списка модулей, трансляции конфигурационных файлов и трансляции файлов исходного кода. Пример выдаваемых частным транслятором сообщений при выполнении этапов трансляции проекта имеет следующий вид:

```
Запуск транслятора.
Проверка входных параметров...
Проверка входных параметров завершена успешно.
Путь к исходному проекту: d:\Translator\TestFiles\Input\ModelDemo.dpr
Путь к результирующему каталогу:
d:\Translator\TestFiles\Output\

Анализ модуля d:\Translator\TestFiles\Input\ModelDemo.dpr
Трансляция конфигурационных фалов проекта завершена

Анализ модуля d:\Translator\TestFiles\Input\ufmMain.pas
Трансляция модуля завершена успешно
Сохранение d:\Translator\TestFiles\Output\ufmMain.cpp
Сохранение d:\Translator\TestFiles\Output\ufmMain.h

Трансляция проекта завершена
```

3. Трансляция проектов при помощи частного транслятора с языка Delphi на язык C++ Builder

Основной целью тестирования частного транслятора с Delphi, разработанного при помощи и на базе Мультитранслятора, являлась проверка его работоспособности и корректности выполняемого перевода программ. Для такого тестирования была выбрана задача из области моделирования электромеханических систем типа «Привод» [8].

Эта система состоит из бензинового двигателя, гибкого вала, однофазного синхронного генератора переменного тока, электрического двигателя и пропеллера. Бензиновый двигатель создает вращательный момент на гибком валу, который, в свою очередь, передает вращательный момент ротору генератора. Генератор вырабатывает переменный ток синусоидальной формы. Электродвигатель, клеммы которого подключены к генератору, создает механический момент, вращающий пропеллер.

В качестве тестовой программы на языке Delphi была написана полная модель системы Привод. Процесс моделирования данной системы представляет собой последовательное вычисление входных и выходных величин для каждого блока через равные интервалы времени.

Для проверки частного транслятора была произведена трансляция модели системы Привод, в результате чего была сгенерирована модель на C++ Builder.

Далее исходная и оттранслированная модели компилировались в соответствующих средах разработки, и выполнялся процесс моделирования. В качестве сравниваемого параметра для двух моделей использовалась зависимость значения силы тока от времени на выходе генератора. На рис. 3 представлена экранная форма результатов моделирования для среды Delphi, а на рис. 4 – для сгенерированной частным транслятором проекта в среде C++ Builder.

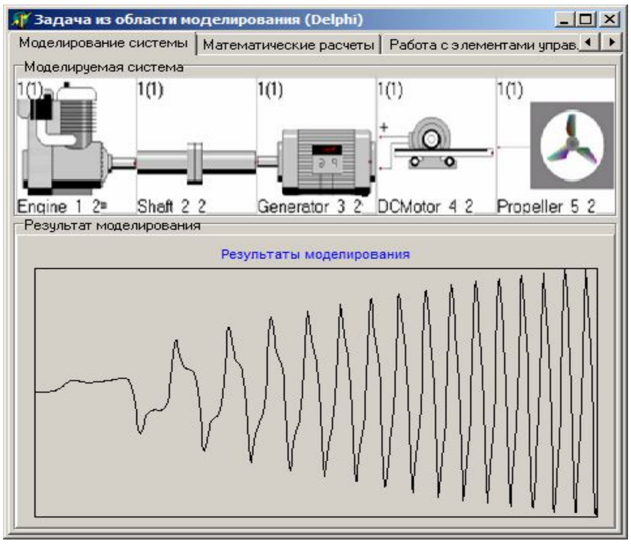


Рисунок 3 – Результаты моделирования исходной программы на Delphi

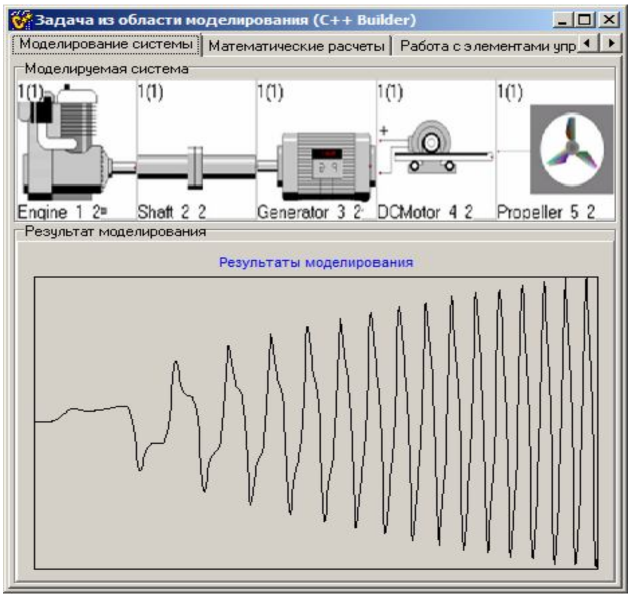


Рисунок 4 – Результаты моделирования оттранслированной частным транслятором проекта в среде C++ Builder

Как следует из рисунков, результаты моделирования переходного процесса совпадают. Это означает, что разработанный частный транслятор *Delphi2C++Builder* корректно осуществляет перевод программы моделирования системы, написанной на языке Delphi, в проект на языке C++ Builder, и поставленная цель построения при помощи Мультитранслятора частного транслятора для пары заданных языков программирования высокого уровня достигнута.

Заключение

Проведенные исследования позволяют сделать вывод о том, что созданная интегрированная инструментальная среда трансляции моделей – Мультитранслятор может быть эффективно использована в режиме компилятора компиляторов для синтеза частных трансляторов с языков программирования высокого уровня.

Для этого режима сохраняется применение таких основных принципов, заложенных в среде Мультитранслятор, как синтез трансляционных (продукционных) модулей, необходимых для частных трансляторов, а также подключение универсального ядра трансляции, используемого как при разработке трансляционных модулей, так и при переводе программных проектов.

Литература

1. Terry P.D. Compilers and Compiler Generators: An Introduction with C++. – International Thomson Computer Press, 1997. – 580 p.
2. List of compiler-compilers. // Wikipedia, the free encyclopedia. – Режим доступа: http://en.wikipedia.org/wiki/List_of_compiler-compilers.
3. Чернухин Ю.В., Поленов М.Ю., Фадеев Р.В. Интерактивная среда мультиязыковой трансляции сложных программных моделей // Анализ и моделирование развивающихся интеллектуальных систем: Межвуз. сб. науч. трудов. – Вып. 4 – Ростов н/Д: Изд-во СКНЦ ВШ, 2003. – С. 10-20.
4. Чернухин Ю.В., Гузик В.Ф., Поленов М.Ю. Инструментальные средства импорта моделей виртуальных моделирующих сред // Искусственный интеллект. – 2006. – № 4. – С. 59-65.
5. Чернухин Ю.В., Гузик В.Ф., Поленов М.Ю. Организация поддержки импорта внешних программных моделей виртуальной среды VTB // Искусственный интеллект. – 2007. – № 4. – С. 497-502.
6. Пашеку Х. Программирование в Borland Delphi 2006 для профессионалов. – М.: Вильямс, 2006. – 944 с.
7. Холингворт Д., Сворт Б., Кэшмэн М., Густавсон П. Borland C++ Builder 6. Руководство разработчика. – М.: Вильямс, 2004. – 976 с.
8. Modelica Models into Multiple Simulation Environments Deploying / Chernukhin Y., Polenov M., Vemulapally C., Solodovnik E., Mantooth A., Dougal R. // Proc. of IEEE International Behavioral Modeling and Simulation Conference (BMAS 2005). – San Jose, CA. – 2005. – P. 134-139.

Yu.V. Chernukhin, M.Yu. Polenov, D.V. Levchenko

Possibilities of the Multitranslation Environment's Application as a Compiler-compiler

The approach to application of previously developed the multilanguage models translation environment – Multitranslator for conversion of program projects from one high level programming languages into projects in other languages is considered. Development of the given approach allows to use the Multitranslator as environment for synthesis of partial translators in the compiler-compiler mode.

Статья поступила в редакцию 22.07.2008.